APPENDIX A

INSTRUCTION SET FOR RISC
PROCESSOR CORE OF IPCM

ADD Addition

Op ration:

```
GReg[r] ← GReg[s] + GReg[r]
T ← (GReg[r] == 0)
```

Assembler
Syntax: `add r,s`

Example: `add 0,3`
to ADD GReg[3] and GReg[0] and store the result in GReg[0]

CPU Flags: T

Cycles: 1

Description: Performs the ADDition of the source General Register s and the destination General Register r, and stores the result in the destination General Register r. The T flag is set if the result of the operation is 0; it is cleared if the result is not zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	0	1	1	s	s	s

Instruction Fields:

`rrr` - destination register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

`sss` - source register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

ADDI — Add with Immediate value

Operation:

```
GReg[r] ← GReg[r] + immediate
T ← (GReg[r] == 0)
```

Assembler

Syntax: `addi r,immediate`

Example: `addi 6,112`
to ADD GReg[6] and decimal value 112 and store the result in GReg[6]

CPU Flags: T

Cycles: 1

Description: Add a zero-extended immediate value to a General Register; stores the result in the General Register. The flag T is set when the result of the operation is zero; otherwise, it is cleared. The immediate value is the low-order byte of the instruction.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:

`000` - GReg[0]
`001` - GReg[1]
`010` - GReg[2]
`011` - GReg[3]
`100` - GReg[4]
`101` - GReg[5]
`110` - GReg[6]
`111` - GReg[7]

`iiiiiiii` - immediate value:

`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

AND Logical AND

Operation:

```
GReg[r] ← GReg[s] & GReg[r]
```

Assembler

Syntax: `and r,s`

Example: `and 1,2`
to AND GReg[1] and GReg[2] and store the result in GReg[1]

CPU Flags: Unaffected

Cycles: 1

Description: Performs the AND of the source General Register s and the destination General Register r, and stores the result in the destination General Register r.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	1	1	1	s	s	s

Instruction Fields:

`rrr` - destination register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

`sss` - source register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

ANDI — Logical AND with Immediate value

Operation:

`GReg[r] ← GReg[r] & immediate`

Assembler

Syntax: `andi r,immediate`

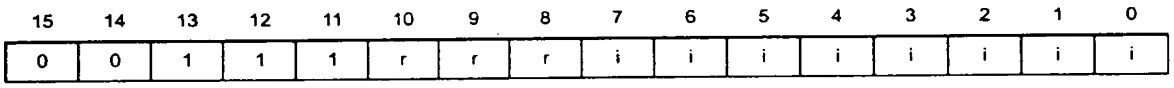
Example: `andi 7,45`
to AND GReg[7] and decimal value 45 and store the result in GReg[7]

CPU Flags: unaffected

Cycles: 1

Description: Performs an AND between a zero-extended immediate value and a General Register; stores the result in the General Register. The immediate value is the low-order byte of the instruction.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - GReg[0]
`001` - GReg[1]
`010` - GReg[2]
`011` - GReg[3]
`100` - GReg[4]
`101` - GReg[5]
`110` - GReg[6]
`111` - GReg[7]

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

ANDN

Logical AND NOT

Operation:

```
GReg[r] ← ~GReg[s] & GReg[r]
```

Assembler

Syntax: `andn r,s`

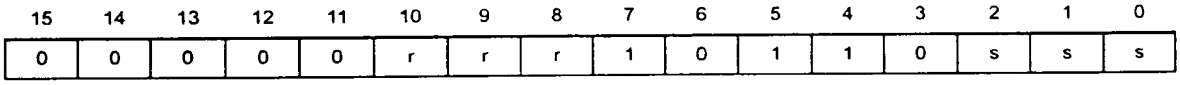
Example: `andn 3,4`
to AND GReg[3] and NOT GReg[4] (bit inverted) and store the result in GReg[3]

CPU Flags: Unaffected

Cycles: 1

Description: Performs the AND of the negation of the source General Register s and the destination General Register r, and stores the result in the destination General Register r.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	1	1	0	s	s	s

Instruction Fields:

`rrr` - destination register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

`sss` - source register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

ANDNI — Logical AND with Negated Immediate value

Operation:

GReg[r] ← GReg[r] & ~immediate

Assembler
Syntax: andni r,immediate

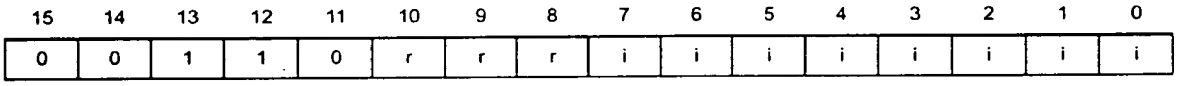
Example: andni 0,2
to AND GReg[0] and decimal value -3 (inverted 32-bit value 2) and store the result in GReg[0]

CPU Flags: unaffected

Cycles: 1

Description: Performs an AND between the negation of a zero-extended immediate value and a General Register; stores the result in the General Register. The immediate value is the low-order byte of the instruction.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

iiiiiiii - immediate value:
00000000 - 0
00000001 - 1
...
11111110 - 254
11111111 - 255

ASR1 Arithmetic Shift Right by 1 Bit

Operation:

`GReg[r]:{b31,b30,...,b1,b0} ← GReg[r]:{b31,b31,b30,...,b1}`

Assembler

Syntax: `asr1 r`

Example: asr1 3
to divide by 2 the signed value of GReg[3] and store the result in GReg[3]

CPU Flags: Unaffected

Cycles: 1

Description: Shift the bits of any General Register to the right and keep the same sign: the left bit (bit 31) is kept untouched.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	1	1	0

Instruction Fields:

`rrr` - register field:

`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

BCLRI

Bit Clear Immediate

Operation:

```
GReg[r]:{b31,...,b(i+1),0,b(i-1),...,b0} ←
GReg[r]:{b31,...,b(i+1),b(i),b(i-1),...,b0}
```

Assembler

Syntax: `bclri r,i`

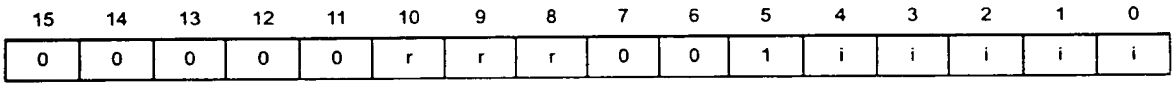
Example: `bclri 1,12`
to clear bit 12 in GReg[1]

CPU Flags: Unaffected

Cycles: 1

Description: Clear the bit of register r specified by the immediate field

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	1	i	i	i	i	i

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

BDF

Conditional Branch if Destination Fault

Operation:

```
if (DF == 1) PC ← PC + 1 + displacement else PC ← PC + 1
```

Assembler

Syntax: `bdf label`

Example: bdf LLL
to jump to LLL if DF is set, or go to the next instruction if DF is cleared; the displacement value is calculated by the assembler

CPU Flags: Unaffected

Cycles: 2 when the branch is done, 1 otherwise

Description: Conditional branch: if flag DF is set, jump to the new address that is calculated by adding the sign-extended 8-bit displacement to the next PC address. If flag DF is cleared, no jump is performed: the next instruction is located at the next PC address.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	p	p	p	p	p	p	p	p

Instruction Fields:

pppppppp - signed displacement field:
00000000 - 0
00000001 - 1
...
01111110 - 126
01111111 - 127
10000000 - -128
10000001 - -127
...
11111110 - -2
11111111 - -1

BF

Conditional Branch if False

Operation:

```
if (T == 0) PC ← PC + 1 + displacement
else PC ← PC + 1
```

Assembler
Syntax: `bf label`

Example: `bf LLL`
to jump to LLL if T is cleared, or go to the next instruction if T is set; the displacement value is calculated by the assembler

CPU Flags: Unaffected

Cycles: 2 when the branch is done, 1 otherwise

Description: Conditional branch: if flag T is cleared, jump to the new address that is calculated by adding the sign-extended 8-bit displacement to the next PC address. If flag T is set, no jump is performed: the next instruction is located at the next PC address.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	0	p	p	p	p	p	p	p	p

Instruction Fields:

pppppppp - signed displacement field:
00000000 - 0
00000001 - 1
...
01111110 - 126
01111111 - 127
10000000 - -128
10000001 - -127
...
11111110 - -2
11111111 - -1

BSETI

Bit Set Immediate

Operation:

```
GReg[r]:{b31,...,b(i+1),1,b(i-1),...,b0} ←
GReg[r]:{b31,...,b(i+1),b(i),b(i-1),...,b0}
```

Assembler

Syntax: `bseti r,i`

Example: bseti 6,5
to set bit 5 in GReg[6]

CPU Flags: Unaffected

Cycles: 1

Description: Sets bit number i in the selected General Register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	1	0	i	i	i	i	i

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

iii - bit number field:
00000 - bit 0
00001 - bit 1
...
11110 - bit 30
11111 - bit 31

BSF — Conditional Branch if Source Fault

Operation:

```
if (SF == 1) PC ← PC + 1 + displacement
else PC ← PC + 1
```

Assembler

Syntax: `bsf label`

Example: `bsf LLL`
to jump to LLL if SF is set, or go to the next instruction if SF is cleared; the displacement value is calculated by the assembler

CPU Flags: Unaffected

Cycles: 2 when the branch is done, 1 otherwise

Description: Conditional branch: if flag SF is set, jump to the new address that is calculated by adding the sign-extended 8-bit displacement to the next PC address. If flag SF is cleared, no jump is performed: the next instruction is located at the next PC address.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0	p	p	p	p	p	p	p	p

Instruction Fields:

pppppppp - signed displacement field:
00000000 - 0
00000001 - 1
...
01111110 - 126
01111111 - 127
10000000 - -128
10000001 - -127
...
11111110 - -2
11111111 - -1

BT Conditional Branch if True

Operation:

```
if (T == 1) PC ← PC + 1 + displacement
else PC ← PC + 1
```

Assembler

Syntax: `bt label`

Example: bt LLL
to jump to LLL if T is set, or go to the next instruction if T is cleared; the displacement value is calculated by the assembler

CPU Flags: Unaffected

Cycles: 2 when the branch is done, 1 otherwise

Description: Conditional branch: if flag T is set, jump to the new address that is calculated by adding the sign-extended 8-bit displacement to the next PC address. If flag T is cleared, no jump is performed: the next instruction is located at the next PC address.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	0	1	p	p	p	p	p	p	p	p

Instruction Fields:

pppppppp - signed displacement field:
00000000 - 0
00000001 - 1
...
01111110 - 126
01111111 - 127
10000000 - -128
10000001 - -127
...
11111110 - -2
11111111 - -1

BTSTI Bit Test immediate

Op ration:

T ← GReg[r]:b(i)

Assembler
Syntax: `btsti r,i`

Example: `btsti 2,29`
to test bit 29 in GReg[2] and copy its value in flag T

CPU Flags: T

Cycles: 1

Description: T is loaded with the value of bit number i from the selected General Register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	1	1	i	i	i	i	i

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

iii - bit number field:
00000 - bit 0
00001 - bit 1
...
11110 - bit 30
11111 - bit 31

CLRF Clear CPU flags

Op ration:

```
if (ff%2 == 0) SF ← 0
if (ff/2 == 0) DF ← 0
```

Assembler
Syntax: `clrf ff`

Example: clrf 2
to clear flag SF and keep flag DF unchanged

CPU Flags: SF, DF

Cycles: 1

Description: Clears a selection of the CPU fault flags: SF, DF, both SF and DF or none can be cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	f	f	0	0	0	0	0	1	1	1

Instruction Fields:

ff - flags field:
00 - clear SF and clear DF
01 - clear DF
10 - clear SF
11 - no clear

CMPEQ — Compare for Equal

Operation:

```
T ← (GReg[s] == GReg[r])
```

Assembler

Syntax: `cmpeq r,s`

Example: `cmpeq 7,5`
to compare GReg[7] and GReg[5] and set flag T if they are equal

CPU Flags: T

Cycles: 1

Description: Substracts the destination General Register r from the source General Register s, and sets T if the result is zero, clears T if the result is not zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	1	0	0	1	s	s	s

Instruction Fields:

`rrr` - destination register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

`sss` - source register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

CMPEQI Compare with Immediate for Equal

Operation:

```
T ← (GReg[r] == immediate)
```

Assembler

Syntax: `cmpeqi r,immediate`

Example: `cmpeqi 2,13`
to compare GReg[2] and decimal value 13 and set flag T if they are equal

CPU Flags: T

Cycles: 1

Description: Substracts the zero-extended immediate value from the General Register, and sets T if the result is zero, clears T if the result is not zero. The immediate value is the low-order byte of the instruction.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

CMPHS — Compare for Higher or Same

Operation:

T ← (GReg[r] ≥ GReg[s])

Assembler
Syntax: `cmphs r,s`

Example: `cmphs 0,1`
to compare GReg[0] and GReg[1] and set flag T if GReg[0] is higher than or equal to GReg[1]

CPU Flags: T

Cycles: 1

Description: Compares the destination General Register r and the source General Register s, and sets T if the destination General Register r is higher than or equal to the source General Register s, clears T otherwise. The comparison is unsigned.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	1	0	1	1	s	s	s

Instruction Fields:

`rrr` - destination register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

`sss` - source register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

CMPLT — Compare for Less Than

Operation:

```
T ← (GReg[r] < GReg[s])
```

Assembler

Syntax: `cmplt r,s`

Example: `cmplt 7,4`
to compare GReg[7] and GReg[4] and set flag T if GReg[7] is lower than GReg[4]

CPU Flags: T

Cycles: 1

Description: Compares the destination General Register r and the source General Register s, and sets T if the destination General Register r is lower than the source General Register s, clears T otherwise. The comparison is signed.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	1	0	1	0	s	s	s

Instruction Fields:

`rrr` - destination register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

`sss` - source register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

DONE DONE, Yield

Op ration:

```
if (jjj&6 == 2) HE[CCR] ← 0
if (jjj == 3) HI[CCR] ← 1
if (jjj == 4) EP[CCR] ← 0
if (jjj&6 == 6) DE[CCR] ← 0
if (jjj == 7) DI[CCR] ← 1
if ((jjj == 0) && (NCP > CCP)) CCR ← NCR
else if ((jjj == 1) && (NCP >= CCP)) CCR ← NCR
else CCR ← NCR
shPC ← {SF,RPC,T,PC}
shLoop ← {LM,EPC,DF,SPC}
shGReg0 ← GReg[0]
(CCR stands for Current Channel Register; NCR stands for Next Channel Register)
```

Assembler

Syntax: done jjj

Example: done 3
to clear HE bit for the current channel, send an interrupt to the Host for the current channel and reschedule

CPU Flags: Unaffected

Cycles: 47 if a context switch is done, 1 otherwise

Description: Clears one of the channel enabling bits (HE, EP or DE for the corresponding channel number) if required, sends an interrupt to the corresponding CPU by setting the appropriate flag if required (HI or DI for the corresponding channel number), and reschedule according to the mode and the NCP (Next Channel Priority) and CCP (Current Channel Priority) values. According to the scheduling decision, the NCR (Next Channel Register) is copied to the CCR (Current Channel Register) and channel contexts are switched.
If several channels with the same highest priority are pending, they are ordered by their number from 31 down to 0: the higher number will be selected (i.e. channel 26 is selected if channels 3, 12, 14 and 26 with the same highest priority are pending).
If no flag is modified, the reschedule can allow the replacement of the current channel by another channel with a priority strictly greater than the current channel priority (yield); or it can allow the replacement of the current channel by another channel with a priority greater than or equal to the current channel priority (yieldge). In the latter case, the selected channel will always be the first one with the same priority, starting from channel number 31 down to channel 0 (the current channel does not belong to the set of selectable channels).

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	j	j	j	0	0	0	0	0	0	0	0

Instruction Fields:

jjj - Channel Flags field:
000 - no channel flags affected: reschedule only if the next channel priority is greater than current channel priority (yield)
001 - no channel flags affected: reschedule only if the next channel priority is greater than

or equal to the current channel priority (yieldge)
010 - clear HE for the current channel and reschedule
011 - clear HE, set HI for the current channel and reschedule
100 - clear EP for the current channel and reschedule
101 - reserved for debug to copy relevant registers into their shadows
110 - clear DE for the current channel and reschedule
111 - clear DE, set DI for the current channel and reschedule

ILLEGAL

ILLEGAL instruction

Operation:

PC ← 0001

Assembler
Syntax: illegal

CPU Flags: Unaffected

Cycles: 2

Description: Jumps to the Illegal instruction routine located at address 0001. All unauthorized instructions result in an Illegal instruction behavior; however, the ILLEGAL instruction must be used to guarantee software compatibility with future versions of the IPCM.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1

Instruction Fields:

JMP — Unconditional Jump Immediate

Operation:

PC ← absolute_address

Assembler
Syntax: `jmp label`

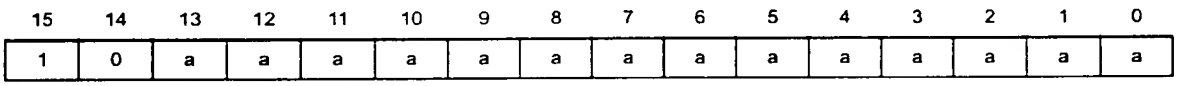
Example: `jmp LLL`
the assembler translates the label to the exact address

CPU Flags: Unaffected

Cycles: 2

Description: Jumps to the absolute address contained the lower 14 bits of the instruction (the PC is a 14-bit register).

Instruction Format:



Instruction Fields:

aaaaaaaaaaaaaa - address field:
00000000000000 - 0
00000000000001 - 1
...
11111111111110 - 16382
11111111111111 - 16383

JMPR

Unconditional Jump

Operation:

PC ← GReg[r]

Assembler

Syntax: `jmpr r`

Example: `jmpr 0`
to jump to address stored in GReg[0]

CPU Flags: Unaffected

Cycles: 2

Description: Jumps to the absolute address contained in a General Register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	0	1	0	0	0

Instruction Fields:

rrr - register field:

000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

JSR — Unconditional Jump to Subroutine Immediate

Operation:

```
RPC ← PC + 1
PC ← absolute_address
```

Assembler
Syntax: `jsrr r`

Example: `jsr LLL`
jumps to subroutine starting at LLL; the assembler translates the label to exact address

CPU Flags: Unaffected

Cycles: 2

Description: Jumps to the subroutine located at the absolute address contained the lower 14 bits of the instruction (the PC is a 14-bit register).

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	a	a	a	a	a	a	a	a	a	a	a	a	a	a

Instruction Fields:

aaaaaaaaaaaaaa - address field:
00000000000000 - 0
00000000000001 - 1
...
11111111111110 - 16382
11111111111111 - 16383

JSRR — Unconditional Jump to Subroutine

Operation:

```
RPC ← PC + 1
PC ← GReg[r]
```

Assembler

Syntax: `jsrr r`

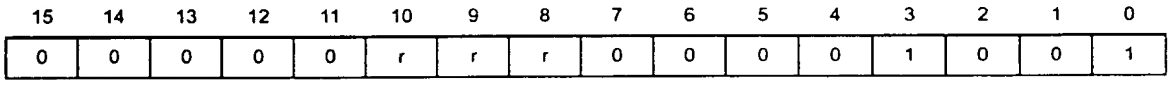
Example: `jsrr 5`
jumps to subroutine located at address stored in GReg[5]

CPU Flags: Unaffected

Cycles: 2

Description: Jumps to the subroutine at address contained in a General Register

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	0	1	0	0	1

Instruction Fields:

`rrr` - register field:

```
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]
```

LD Load Register

Operation:

```
GReg[r] ← [GReg[b] + displacement]
if (transfer_error) SF ← 1
else SF ← 0
```

Assembler
Syntax: `ld r,(b,displacement)`

Example: `ld 1,(2,23)`
loads data into GReg[1]; the data is located at address obtained by adding decimal value 23 to GReg[2]

CPU Flags: SF

Cycles: 2+n where n is 0 for ROM, RAM or memory mapped registers, and n is the number of wait-states of the peripheral for a peripheral access

Description: Adds a 5-bit zero-extended displacement to a base address in General Register b; the result is the address of the data to fetch on the DM bus. The data received from the bus is stored in the destination General Register r. If an error occurs during the transfer, the flag SF is set, else it is cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	r	r	r	d	d	d	d	d	b	b	b

Instruction Fields:

rrr - destination register field:
000 - GReg[0]
001 - GReg[1]
...
111 - GReg[7]

bbb - base address register field:
000 - GReg[0]
001 - GReg[1]
...
110 - GReg[6]
111 - GReg[7]

ddddd - displacement value:
00000 - 0
00001 - 1
...
11110 - 30
11111 - 31

LDF — Load Register from Functional Unit

Operation:

```
GReg[r] ← [fu_address]
if (transfer_error) SF ← 1
else SF ← 0
fu_address is an 8-bit field:
                7:3    fureg
                2      fetch
                1:0    size
```

Assembler
Syntax: `ldf r,fu_address`

Example: `ldf 0,18`
loads data coming from the Host DMA register MD into GReg[0]; it is a 16-bit access with no prefetch

CPU Flags: SF

Cycles: 1+n where n is the number of wait-states that may be inserted by the functional unit

Description: Sends an 8-bit address on the Functional Unit Bus (FU bus) and stores the data received from the bus in the destination General Register r. If an error occurs during the transfer, the flag SF is set, else it is cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	r	r	r	u	u	u	u	u	u	u	u

Instruction Fields:

`rrr` - destination register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
...
`110` - `GReg[6]`
`111` - `GReg[7]`

uuuuuuuu - functional unit address field (x is a don't-care bit):
`00000xxx`- read MA (no side effect)
`00010x00`- read MD (no side effect)
`00010p01`- read MD: 8-bit access
`MA ← MA + 1`
`prefetch if ((p == 1) && (MA%4 == 0))`
`00010p10`- read MD: 16-bit access
`MA ← MA + 2`
`prefetch if ((p == 1) && (MA%4 == 0))`
`00010p11`- read MD: 32-bit access
`MA ← MA + 4`

```
prefetch if ((p == 1) && (MA%4 == 0))
```
00011xxx- read MS (no side effect)
00100xxx- read DA (no side effect)
00110x00- read DD (no side effect)
00110p01- read DD: 8-bit access
```
DA ← DA+1
prefetch if ((p == 1) && (DA%4 == 0))
```
00110p10- read DD: 16-bit access
```
DA ← DA+2
prefetch if ((p == 1) && (DA%4 == 0))
```
00110p11- read DD: 32-bit access
```
DA ← DA+4
prefetch if ((p == 1) && (DA%4 == 0))
```
00111xxx- read DS (no side effect)
01000xxx- read CA (no side effect)
01001xxx- read CS (right aligned, no side effect)

LDI

Load Register with Immediate value

Operation:

GReg[r] ← immediate

Assembler
Syntax: `ldi r,immediate`

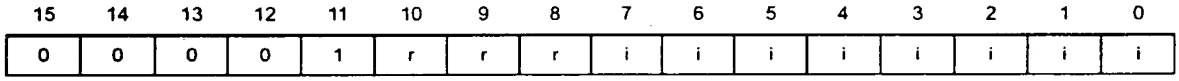
Example: `ldi 6,1`
loads decimal value 1 into GReg[6]

CPU Flags: Unaffected

Cycles: 1

Description: Stores a zero-extended immediate value in a General Register. The immediate value is the low-order byte of the instruction.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - GReg[0]
`001` - GReg[1]
`010` - GReg[2]
`011` - GReg[3]
`100` - GReg[4]
`101` - GReg[5]
`110` - GReg[6]
`111` - GReg[7]

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

LDRPC — Load from RPC to Register

Operation:

GReg[r] ← RPC

Assembler
Syntax: `ldrpc r`

Example: `ldrpc 3`
copies RPC to GReg[3]

CPU Flags: Unaffected

Cycles: 1

Description: Stores the contents of the RPC in a General Register. That instruction may be used to have more than one level of subroutines.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	0	1	0	1	0

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

LOOP Hardware Loop

Operation:

```
if (ff%2 == 0) SF ← 0
if (ff/2 == 0) DF ← 0
if ((GReg[0] == 0) || (SF == 1) || (DF == 1))
                PC ← PC + loop_size + 1
else {
                SPC ← PC + 1
                EPC ← PC + loop_size + 1
                LM ← 1
                PC ← PC + 1
}
during each instruction execution in the loop:
if ((SF == 1) || (DF == 1)) {
                LM ← 0
                PC ← EPC
}
else if ((PC + 1) == EPC) {
                GReg[0] ← GReg[0] - 1
                if (GReg[0] == 0) {
                        LM ← 0
                        PC ← EPC
                }
                else PC ← SPC
}
else PC ← nextPC(instruction)
after the execution of the last instruction of the loop body:
if (GReg[0] == 0)
                T ← 1
else
                T ← 0
```

Assembler

Syntax: `loop n{,ff}`

Example: `loop 3,0`
executes GReg[0] times the instructions comprised between PC+1 and PC+3 (included); both SF and DF flags are cleared before starting the loop; when omitted, the `ff` field will be set to 0 (clearing both SF and DF)

CPU Flags: LM, T

Cycles: 2 when the loop count (GReg[0]) is 0 or SF or DF is set at loop start, 1+1 when the loop starts but exits abnormally (SF or DF set inside the loop which adds 1 cycle to the offending load or store to jump to EPC), 1 when the loop is executed normally

Description: The LOOP instruction executes several times a sequence of instructions. The number of times is given by the contents of GReg[0] that is the loop counter. That means the IPCM will jump to the first instruction after the end of the loop if GReg[0] value is 0; if not, the IPCM enters loop mode: it sets the LM flag that will only be reset once the last instruction of the last loop is executed. The instructions in the loop will be executed GReg[0] times.
The management of fault flags (SF and DF) is as follows: when entering the hardware loop, SF and DF can be cleared according to the ff field of the instruction; after that operation, if any flag is still set, the loop will not be executed: the IPCM will jump to the first instruction after the end of the loop without entering loop mode. During the execution of the loop, if any fault flag is set by a LD, LDF, ST or STF instruction, the IPCM will immediately exit loop mode and jump to the first instruction after the end of the loop. In that case, GReg0 is not decremented for that last piece of the loop body execution (this is even the case if the SF or DF flag is set at the last instruction of the loop body).
The T flag reflects the state of GReg[0] after the end of the loop, which is an indicator of the complete

execution of the loop: if the loop exited because of an error (SF or DF set), GReg[0] will not be 0 at the end of the loop, hence T will be cleared; if the loop execution went well, GReg[0] will be 0 at the end of the loop, hence T will be set. The boundary case when a source or destination fault occurs at the last instruction of the last loop is considered as an anticipated exit of the loop, which causes the T flag to be cleared. If the last instruction executed before leaving the hardware loop also tries to modify the T flag, the flag is updated according to the value of GReg[0], NOT according to the result of the last executed instruction.

Limitations: Jump instructions (JMP, JMPR, JSR, JSRR, BF, BT, BSF, BDF) are not allowed inside the hardware loop (we are working on this: some jumps will be allowed in the future but beware of boundary cases - the exact behavior of the hardware will be completely specified in all the cases).
- GReg[0] cannot be written to inside the hardware loop (it can be read).
- the empty loop (0 instruction in the body) is forbidden.
- if GReg[0] == 0 at the start of the loop, which causes a jump to EPC, the T flag is not updated (we are also working on this: the intention is to have the T flag set).

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	0	f	f	n	n	n	n	n	n	n	n

Instruction Fields:

ff - flags field:
00 - clear SF and clear DF
01 - clear DF
10 - clear SF
11 - no clear

nnnnnnnn - loop size
00000000 - empty loop: forbidden value
00000001 - 1 instruction in the loop
00000010 - 2 instructions in the loop
...
11111111 - 255 instructions in the loop

LSL1 Logical Shift Left by 1 Bit

Operation:

`GReg[r]:{b30,...,b1,b0,0} ← GReg[r]:{b31,b30,...,b1,b0}`

Assembler

Syntax: `lsl1 r`

Example: `lsl1 2`
multiplies by 2 the value in GReg[2]

CPU Flags: Unaffected

Cycles: 1

Description: Shift the bits of any General Register to the left. The right bit (bit 0) is set to 0. No overflow is detected by the hardware.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	1	1	1

Instruction Fields:

`rrr` - register field:

`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

LSR1 Logical Shift Right by 1 Bit

Op ration:

`GReg[r]:{0,b31,b30,...,b1}` ← `GReg[r]:{b31,b30,...,b1,b0}`

`Syntax:` `lsr1 r`

Example: `lsr1 4`
divides by 2 the unsigned value contained in GReg[4]

CPU Flags: Unaffected

Cycles: 1

Description: Shift the bits of any General Register to the right. The left bit (bit 31) is set to 0.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	1	0	1

Instruction Fields:

`rrr` - register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

MOV

Logical Move

Operation:

GReg[r] ← GReg[s]

Assembler

Syntax: `mov r,s`

Example: `mov 4,0`
copies GReg[0] to GReg[4]

CPU Flags: Unaffected

Cycles: 1

Description: Move the contents of the source General Register s to the destination General Register r.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	0	0	1	s	s	s

Instruction Fields:

rrr - destination register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

sss - source register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

NOTIFY

Notify to MCU/DSP

Op ration:

```
if (jjj&4 == 0) {
                if (jjj&2 == 2) HE[CCR] ← 0
                if (jjj&1 == 1) HI[CCR] ← 1
}
else if (jjj == 4) EP[CCR] ← 0
else {
                if (jjj&2 == 2) DE[CCR] ← 0
                if (jjj&1 == 1) DI[CCR] ← 1
}
(CCR stands for Current Channel Register)
```

Assembler

Syntax: notify jjj

Example: notify 7
clears the DE bit for the current channel and sends an interrupt to the DSP for the current channel

CPU Flags: Unaffected

Cycles: 1

Description: Clears one of the channel enabling bits (HE, EP or DE for the corresponding channel number) if required, sends an interrupt to the corresponding CPU by setting the appropriate flag if required (HI or DI for the corresponding channel number).

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	j	j	j	0	0	0	0	0	0	0	1

Instruction Fields:

jjj - Channel Flags field:
000 - unused
001 - set HI for the current channel
010 - clear HE for the current channel
011 - clear HE, set HI for the current channel
100 - clear EP for the current channel
101 - set DI for the current channel
110 - clear DE for the current channel
111 - clear DE, set DI for the current channel

OR Logical OR

Operation:

```
GReg[r] ← GReg[s] | GReg[r]
```

Assembler
Syntax: `or r,s`

Example: `or 3,6`
ORs GReg[3] and GReg[6] and stores the result in GReg[6]

CPU Flags: Unaffected

Cycles: 1

Description: Performs the OR of the source General Register s and the destination General Register r, and stores the result in the destination General Register r.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	1	0	1	s	s	s

Instruction Fields:

`rrr` - destination register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

`sss` - source register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

ORI Logical OR with Immediate value

Operation:

```
GReg[r] ← GReg[r] | immediate
```

Assembler

Syntax: `ori r,immediate`

Example: `ori 1,56`
ORs GReg[1] and the decimal value 56 and stores the result in GReg[1]

CPU Flags: unaffected

Cycles: 1

Description: Performs an OR between a zero-extended immediate value and a General Register; stores the result in the General Register. The immediate value is the low-order byte of the instruction.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

RET Return from subroutine

Operation:

PC ← RPC

Assembler
Syntax: ret

CPU Flags: Unaffected

Cycles: 2

Description: Return from subroutine.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Instruction Fields:

REVB

Reverse Byte order

Operation:

GReg[r]:{B3,B2,B1,B0} ← GReg[r]:{B0,B1,B2,B3}

Assembler

Syntax: `revb r`

Example: `revb 5`
reverses bytes order in GReg[5]

CPU Flags: Unaffected

Cycles: 1

Description: Reverse the byte order of any General Register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	0	0	0

Instruction Fields:

`rrr` - register field:

000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

REVBLO Reverse Low Order Bytes

Operation:

GReg[r]:{B3,B2,B0,B1} ← GReg[r]:{B3,B2,B1,B0}

Assembler

Syntax: `revblo r`

Example: `revblo 0`
reverses low order bytes in GReg[0]

CPU Flags: Unaffected

Cycles: 1

Description: Reverse both low order bytes of any General Register.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	0	0	1

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

ROR1

Rotate Right by 1 bit

Op ration:

`GReg[r]:{b0,b31,b30,...,b1}` ← `GReg[r]:{b31,b30,...,b1,b0}`

Assembler

Syntax: `rorl r`

Example: `rorl 3`
rotates bits to the right in GReg[3]

CPU Flags: Unaffected

Cycles: 1

Description: Rotate the bits of any General Register to the right.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	1	0	0

Instruction Fields:

`rrr` - register field:

`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

RORB

Rotate Right by 1 Byte

Operation:

`GReg[r]:{B0,B3,B2,B1}` ← `GReg[r]:{B3,B2,B1,B0}`

Assembler
Syntax: `rorb r`

Example: `rorb 2`
rotates bytes to the right in GReg[2]

CPU Flags: Unaffected

Cycles: 1

Description: Rotate the bytes of any General Register to the right.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	0	0	0	1	0	0	1	0

Instruction Fields:

`rrr` - register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

SOFTBKPT Software Breakpoint

Operation:

Stops the current script and enters debug mode

Assembler
Syntax: softbkpt

CPU Flags: Unaffected

Cycles:

Description:

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Instruction Fields:

ST Store Register

Op ration:

```
[GReg[b] + displacement] ← GReg[r]
if (transfer_error) DF ← 1
else DF ← 0
```

Assembler

Syntax: `st r,(b,displacement)`

Example: `st 7,(0,9)`
stores the value from GReg[7] into memory at address obtained by adding decimal value 9 to GReg[0]

CPU Flags: DF

Cycles: 2+n where n is 0 for ROM, RAM or memory mapped registers, and n is the number of wait-states of the peripheral for a peripheral access

Description: Adds a 5-bit zero-extended displacement to a base address in General Register b; the result is the address of the data to store on the DM bus. The data sent on the bus comes from the source General Register r. If an error occurs during the transfer, the flag DF is set, else it is cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	r	r	r	d	d	d	d	d	b	b	b

Instruction Fields:

rrr - source register field:
000 - GReg[0]
001 - GReg[1]
...
110 - GReg[6]
111 - GReg[7]

bbb - base address register field:
000 - GReg[0]
001 - GReg[1]
...
110 - GReg[6]
111 - GReg[7]

ddddd - displacement value:
00000 - 0
00001 - 1
...
11110 - 30
11111 - 31

STF

Store Register in Functional Unit

Operation:

```
[fu_address] ← GReg[r]
if (transfer_error) DF ← 1
else DF ← 0
fu_address is an 8-bit field:
              7:3    fureg
              2      fetch / flush
              1:0    size
```

Assembler
Syntax: `stf r,fu_address`

Example: `stf 3,55`
stores the 32-bit contents of GReg[3] to the DSP DMA register DD; waits until the flush to external DSP memory is completed

CPU Flags: DF

Cycles: 1+n where n is the number of wait-states that may be inserted by the functional unit

Description: Sends an 8-bit address on the Functional Unit Bus (FU bus) and sends the contents of the source General Register r on the bus. If an error occurs during the transfer, the flag DF is set, else it is cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	r	r	r	u	u	u	u	u	u	u	u

Instruction Fields:

rrr - source register field:
000 - GReg[0]
001 - GReg[1]
...
110 - GReg[6]
111 - GReg[7]

uuuuuuuu - functional unit address field (x is a don't-care bit):
00000x00- write MA (no side effect)
00000p01- write MA
10flush if (MD not empty)
11prefetch if (p == 1)
transfer_error if ((p ==1) && (MD not empty))
00010000- write MD (no side effect)
00010100- no write: flush if MD is not empty
00010f01- write MD: 8-bit access
MA ← MA + 1
flush if ((f == 1) || (MA%4 == 0))

00010f10- write MD: 16-bit access
MA ← MA + 2
flush if ((f == 1) || (MA%4 == 0))
00010f11- write MD: 32-bit access
MA ← MA + 4
flush if ((f == 1) || (MA%4 == 0))
00011xxx- write MS (no side effect)
00100x00- write DA (no side effect)
00100p01- write DA
10flush if (DD not empty)
11prefetch if (p == 1)
00110000- write DD (no side effect)
00110100- no write: flush if DD is not empty
00110f01- write DD: 8-bit access
DA ← DA+1
flush if ((f == 1) || (DA%4 == 0))
00110f10- write DD: 16-bit access
DA ← DA+2
flush if ((f == 1) && (DA%4 == 0))
00110f11- write DD: 32-bit access
DA ← DA+4
flush if ((f == 1) && (DA%4 == 0))
00111xxx- write DS (no side effect)
01000xxx- write CA (no side effect)
01001xx0- write CS (right aligned, no side effect)
01001xx1- write CS: compute CRC with new incoming byte

SUB Substract

Operation:

```
GReg[r] ← GReg[r] - GReg[s]
T ← (GReg[r] == 0)
```

Assembler

Syntax: `sub r,s`

Example: `sub 4,7`
SUBstracts GReg[7] from GReg[4] and stores the result in GReg[4]

CPU Flags: T

Cycles: 1

Description: Substracts the source General Register s from the destination General Register r, and stores the result in the destination General Register r. The T flag is set if the result of the operation is 0; it is cleared if the result is not zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	1	0	0	s	s	s

Instruction Fields:

rrr - destination register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

sss - source register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

SUBI

Substract with Immediate

Operation:

`GReg[r] ← GReg[r] - immediate T ← (GReg[r] == 0)`

Assembler

Syntax: `sub r,immediate`

Example: `sub 1,255`
SUBstracts decimal value 255 from GReg[1] and stores the result in GReg[1]

CPU Flags: T

Cycles: 1

Description: Substracts a zero-extended immediate value from a General Register; stores the result in the General Register. The flag T is set when the result of the operation is zero; otherwise, it is cleared. The immediate value is the low-order byte of the instruction.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - GReg[0]
`001` - GReg[1]
`010` - GReg[2]
`011` - GReg[3]
`100` - GReg[4]
`101` - GReg[5]
`110` - GReg[6]
`111` - GReg[7]

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

TST Test with Zero

Operation:

```
T ← ((GReg[s] & GReg[r]) != 0)
```

Assembler
Syntax: `tst r,s`

Example: `tst 2,3`
ANDs GReg[2] and GReg[3] and sets T if the result is non-null

CPU Flags: T

Cycles: 1

Description: Performs the AND of the source General Register s and the destination General Register r, and sets T if the result is not zero, clears T if the result is zero.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	1	0	0	0	s	s	s

Instruction Fields:

`rrr` - destination register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

`sss` - source register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

TSTI

Mask with Zero Immediate

Operation:

```
T ← ((GReg[r] & immediate) != 0)
```

Assembler
Syntax: `tsti r,immediate`

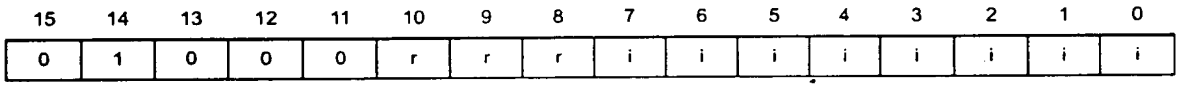
Example: `tsti 5,13`
ANDs GReg[5] and decimal value 13 and sets T if the result is non-null

CPU Flags: T

Cycles: 1

Description: Performs the AND of a zero-extended immediate value and the destination General Register r, and sets T if the result is not zero, clears T if the result is zero. The immediate value is the low-order byte of the instruction.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

`rrr` - register field:
`000` - GReg[0]
`001` - GReg[1]
`010` - GReg[2]
`011` - GReg[3]
`100` - GReg[4]
`101` - GReg[5]
`110` - GReg[6]
`111` - GReg[7]

`iiiiiiii` - immediate value:
`00000000` - 0
`00000001` - 1
...
`11111110` - 254
`11111111` - 255

XOR Logical Exclusive OR

Op ration:

`GReg[r] ← GReg[s] ^ GReg[r]`

Assembler
Syntax: `xor r,s`

Example: `xor 0,3`
XORs GReg[0] and GReg[3] and stores the result in GReg[0]

CPU Flags: Unaffected

Cycles: 1

Description: Performs the eXclusive OR of the source General Register s and the destination General Register r, and stores the result in the destination General Register r.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	r	r	r	1	0	0	1	0	s	s	s

Instruction Fields:

`rrr` - destination register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

`sss` - source register field:
`000` - `GReg[0]`
`001` - `GReg[1]`
`010` - `GReg[2]`
`011` - `GReg[3]`
`100` - `GReg[4]`
`101` - `GReg[5]`
`110` - `GReg[6]`
`111` - `GReg[7]`

XORI

Exclusive OR with Immediate

Operation:

```
GReg[r] ← GReg[r] ^ immediate
```

Assembler

Syntax: `xori r,immediate`

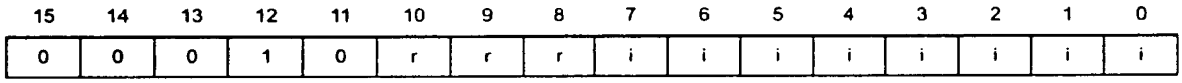
Example: `xor 7,5`
XORs GReg[5] and decimal value 5 and stores the result in GReg[7]

CPU Flags: Unaffected

Cycles: 1

Description: Performs an eXclusive OR between a zero-extended immediate value and a General Register; stores the result in the General Register. The immediate value is the low-order byte of the instruction.

Instruction Format:



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	r	r	r	i	i	i	i	i	i	i	i

Instruction Fields:

rrr - register field:
000 - GReg[0]
001 - GReg[1]
010 - GReg[2]
011 - GReg[3]
100 - GReg[4]
101 - GReg[5]
110 - GReg[6]
111 - GReg[7]

iiiiiiii - immediate value:
00000000 - 0
00000001 - 1
...
11111110 - 254
11111111 - 255

MvShPC2Gr1 Move Data from Shadow PC register to Register 1

Op ration:

GReg[1] ← ShPCReg

Assembler
Syntax: none as this instruction should only be used through the OnCE

CPU Flags: Unaffected

Cycles: 1

Description: Once debug specific instruction. Move the contents of the shadow PC register to the General register[1].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Instruction Fields:

MvShLoop2Gr1 Move Data from Shadow Loop register to Register 1

Operation:

GReg[1] ← ShLoopReg

Assembler Syntax: none as this instruction should only be used through the OnCE

CPU Flags: Unaffected

Cycles: 1

Description: Once debug specific instruction. Move the contents of the shadow Loop register to the General register[1].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

Instruction Fields:

MvShGr02Gr1 Move Data from Shadow GReg[0] register to Register 1

Operation:

GReg[1] ← ShGReg[0]

Assembler

Syntax: none as this instruction should only be used through the OnCE

CPU Flags: Unaffected

Cycles: 1

Description: Once debug specific instruction. Move the contents of the shadow GReg[0] register to the General register[1 The ShGReg[0] register is used during context switch.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0

Instruction Fields:

MvGr12ShPC Move Data from Register 1 to Shadow PC register

Operation:

ShPCReg ← GReg[1]

Assembler
Syntax: none as this instruction should only be used through the OnCE

CPU Flags: Unaffected

Cycles: 1

Description: Once debug specific instruction. Move the contents of the General register[1] to the Shadow PC register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0

Instruction Fields:

MvGr12ShLoop Move Data from Register 1 to Shadow Loop register

Operation:

ShPCReg ← GReg[1]

Assembler
Syntax: none as this instruction should only be used through the OnCE

CPU Flags: Unaffected

Cycles: 1

Description: Once debug specific instruction. Move the contents of the General register[1] to the Shadow Loop register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Instruction Fields:

reschedule

Assembler
Syntax: `reschedule`

CPU Flags: Unaffected

Cycles: 1

Description: Depending on HPPR and HPCR (TestPending), the instruction will either put the ipcm core in IDLE mode or continue the context switch subroutine (AndSwitch).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0

Instruction Fields:

CtxPtrInit --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

GReg[0] ← DM[0x7002]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: Unaffected

Cycles: 1

Description: The base address where all control/general registers will be spilled on execution of a context switch instruction (done or yield) is stored in GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1

Instruction Fields:

CatchCPtr--Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

]

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: Unaffected

Cycles: 1

Description: The base address where all control/general registers will be spilled on execution of a context switch instruction (done or yield) is stored in GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	1

Instruction Fields:

ldMAstG1 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[1]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
MA -> GReg[1]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: T

Cycles: 1

Description: store GReg[1] to memory, update GReg[1] with MA and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0

Instruction Fields:

ldMDstG2--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[2]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
MD -> GReg[2]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: T

Cycles: 1

Description: store GReg[2] to memory, update GReg[2] with MD and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0

Instruction Fields:

ldMSstG3--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[3]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
MS -> GReg[3]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: store GReg[3] to memory, update GReg[3] with MS and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0

Instruction Fields:

ldDAstG4--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[4]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
DA-> GReg[4]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: store GReg[4] to memory, update GReg[4] with DA and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0

Instruction Fields:

ldDDstG5--Context Switch specific instruction--

= THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE=

Operation:

```
GReg[5]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
DD-> GReg[5]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: store GReg[5] to memory, update GReg[5] with DD and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0

Instruction Fields:

ldDSstG6--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[6]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
DS-> GReg[6]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: store GReg[6] to memory, update GReg[6] with DD and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0

Instruction Fields:

LdCAstG7 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[7]->DM[ G[0] + 1 ]
G[0] + 1 -> G[0]
CA-> GReg[7]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: T

Cycles: 1

Description: store GReg[7] to memory, update GReg[7] with CA and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0

Instruction Fields:

stG7mvShPC --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[7]->DM[ G[0] + 1]
G[0] + 1 -> G[0]
ShPCReg -> GReg[7]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: T

Cycles: 1

Description: store GReg[7] to memory, update GReg[7] with ShPCReg and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1

Instruction Fields:

stG7mvShLoop--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[7]->DM[ G[0] + 1]
G[0] + 1 -> G[0]
ShLoopReg -> GReg[7]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: T

Cycles: 1

Description: store GReg[7] to memory, update GReg[7] with ShLoopReg and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1

Instruction Fields:

stG7ldCS--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[7]->DM[ G[0] + 1]
G[0] + 1 -> G[0]
CS -> GReg[7]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: store GReg[7] to memory, update GReg[7] with CS and increment address pointer GReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0

Instruction Fields:

stCAmovShReg02Gr1 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[7]->DM[ G[0] + 1]
ShReg0 -> GReg[1]
```

Assembler

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: none

Cycles: 1

Description: store GReg[7] to memory, update GReg[1] with ShReg0.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0

Instruction Fields:

TstPendingAndSwitch--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Syntax: `none as this instruction can not be used outside of the ROM context switch routine.`

CPU Flags: none1

Description: Depending on HPPR and HPCR (TestPending), the instruction will either put the ipcm core in IDLE mode or continue the context switch subroutine (AndSwitch).
During same cycle, content of GeneralReg[1] will be stored in Data Memory at address pointed by GeneralReg[0].

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0

Instruction Fields:

ldFU0inLd0 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
read access DM[G[0]]
G[0] + 1 -> G[0]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: Start a read access at GReg[0] address and increments address pointer GReg[0]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	1	1

Instruction Fields:

mvFU02G1 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	1	1

Instruction Fields:

ldmfub7 --Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

GReg[1] -> MA

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub6--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

GReg[1] -> MD

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub5 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

GReg[1] -> MS

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub4--Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

GReg[1] -> DA

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub3 --Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Op ration:

GReg[1] -> DD

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub2 --Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

```
GReg[1] -> DS
(Soft pipeline) result of former cycle initiated read -> GReg[1]
(Soft pipeline) initiate DM[ G[0] ] read access. Data available next cycle.
GReg[0] - 1 -> GReg[0]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub1 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

GReg[1] -> CA

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1

Instruction Fields:

ldmfub0 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

```
GReg[1] -> CS
(Soft pipeline) result of former cycle initiated read -> GReg[1]
(Soft pipeline) initiate DM[ G[0] ] read access. Data available next cycle.
GReg[0] - 1 -> GReg[0]
```

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1

Instruction Fields:

ldShLoop--Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> ShLoop

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	1

Instruction Fields:

ldShPC --Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> ShPC

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	1

Instruction Fields:

ldmGReg7 --Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[7]

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	0

Instruction Fields:

IdmGReg6--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[6]

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	0

Instruction Fields:

ldmGReg5--Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[5]

GReg[0] - 1 -> GReg[0]

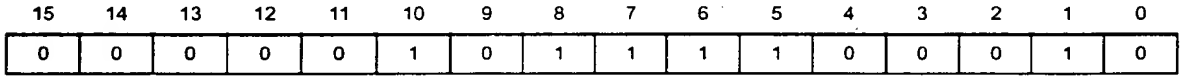
Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	1	1	1	0	0	0	1	0

Instruction Fields:

ldmGReg4--Context Switch specific instruction--

– THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE–

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[4]

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0

Instruction Fields:

ldmGReg3--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[1]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[3]

GReg[0] - 1 -> GReg[0]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	0

Instruction Fields:

ldmGReg2--Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[0]

(Soft pipeline) initiate DM[G[0]] read access. Data available next cycle.

GReg[1] -> GReg[2]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: T

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0

Instruction Fields:

IdmGReg1GReg0 --Context Switch specific instruction--

-- THIS INSTRUCTION CAN NOT BE USED OUTSIDE OF THE ROM-CONTEXT SWITCH ROUTINE--

Operation:

(Soft pipeline) result of former cycle initiated read -> GReg[0]

GReg[0] -> GReg[1]

Assembler

Syntax: none as this instruction can not be used outside of the ROM context switch routine.

CPU Flags: none

Cycles: 1

Description: .

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0

Instruction Fields:

cpShReg

Assembl r
Syntax: cpShReg.

CPU Flags: none

Cycles: 1

Description: SF, RPC, T, PC registers are updated according to the value of their corresponding bits in the ShPC register.LM, EPC, DF, SPC registers are updated according to the value of their corresponding bits in the ShLoop register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0

Instruction Fields: